DECLARATIVE MARKUP FOR SCORING MULTIPLE TIME-BASED ASSETS AND EVENTS WITHIN A SCENE COMPOSITION SYSTEM

Christopher F. Marrin
James R. Kent
Robert K. Myers
Peter G. Broadwell

RELATED APPLICATION

The present application claims priority from provisional patent application Ser. No. 60/146,972, filed on August 3, 1999, now pending.

BACKGROUND OF THE INVENTION

Field of the Invention

This invention relates generally to modeling language for 3D graphics and, more particularly, to temporal manipulation of media assets.

Description of the Related Prior Art

Conventional modeling languages for real-time 3D scene rendering have traditionally focussed on aspects of scene structure, geometry, appearance, and, to some degree, animation, and interactivity. This focus has been driven by the following two factors. First, 3D computer graphics applications have been geared toward user-driven experiences and, thus, tend to be structured around a rendered response to events. Second, the majority of these applications take a "render it as fast as you can" approach to scene updates, with little respect paid to fidelity of the time base. Conventional modeling languages fail to provide the accuracy of temporal relationship between two media assets. For example, if a video asset and an audio asset are to start at the same time, this can be achieved by prescribing start time for each asset independent of other assets. This allows the start times to be slightly different. It is desirable that th start time for each asset be controlled by the same field, thereby

resulting in accurate synchronization of the assets. Media assets include audio media, video media, animations, audio-visual media, images or events.

As full motion video and high fidelity audio are integrated into a scene rendering mix, it is desirable to deliver high quality television-like viewing experiences while supporting viewer interactivity. It is desirable to provide a passive viewing experience that is more television-like and not a web page-like viewing experience.

In a declarative markup language, the semantics required to attain the desired outcome are implicit in the description of the outcome. It is not necessary to provide a separate procedure (i.e., write a script) to get the desired outcome. One example of a declarative language is HyperText Markup Language (HTML).

Various approaches to scoring animation and playback have previously been developed in other computer-based media, including Macromedia Director and the W3C's Synchronized Multimedia Integration Language (SMIL). However, these existing scoring systems do not allow for declarative composition of a real-time scene wherein the independent scores are dynamically composed and decomposed hierarchically, structuring time in manner akin to the spatial scene graph. For example structuring blocks of time to be next to each other or structuring block of time to be parallel (synchronized) with each other. The conventional scoring systems do not allow variable rate and direction of score evaluation to be done declaratively, and neither do they allow declarative implementation of a modular computation strategy based upon a normalized "fraction done" output, suitable for rapid assembly and reuse of behavioral animation.

SUMMARY OF THE INVENTION

A system and method for declarative markup that allows temporal manipulation of media assets is presented. The media assets can be audio media, video media, animations, audio-visual media, images or events. Using the present invention a media sequence can be formed by playing more than one medium in series, in parallel or in any other temporal combination wherein a medium is cued to another medium. A media sequence created using the present invention can become part of a new media sequence, and the rate of playing the

media sequence can be controlled by fields associated with the new media sequence. Also, using present invention, a media sequence can be cued to start playing at a fixed time before the end of a first media sequence, and in this instance the length of the first media sequence can be varied while still maintaining the fixed time from the end of the first media sequence.

BRIEF DESCRIPTION OF THE DRAWING

Fig. 1A shows the basic architecture of Blendo.

Fig. 1B is a flow diagram illustrating flow of content through Blendo engine.

Fig. 2A shows time relationship between media sequences in a score.

Fig. 2B illustrates synchronization of the media sequence of Fig. 2 requiring pre-loading

Fig. 3 shows time relationships between various constituent media of an interactive presentation.

DETAILED DESCRIPTION

Blendo is an exemplary embodiment of the present invention that allows temporal manipulation of media assets including control of animation and visible imagery, and cueing of audio media, video media, animation and event data to a media asset that is being played. Fig. 1A shows basic Blendo architecture. A comprehensive description of Blendo can be found in Apendix A. At the core of the Blendo architecture is a Core Runtime module 10 (Core hereafter) which presents various Application Programmer Interface (API hereafter) elements and the object model to a set of objects present in system 11. During normal operation, a file is parsed by parser 14 into a raw scene graph 16 and passed on to Core 10, where its objects are instantiated and a runtime scene graph is built. The objects can be built-in objects 18, author defined objects 20, native objects 24, or the like. The objects use a set of available managers 26 to obtain platform services 32. These platform services 32 include event handling, loading of assets, playing of media, and the like. The objects use rendering layer 28 to compose intermediate or final images for display. A page integration component 30 is used to interface Blendo to an external environment, such as an HTML or XML page.

Blendo contains a system object with references to the set of managers 26. Each manager 26 provides the set of APIs to control some aspect of system 11. An event manager 26D provides access to incoming system events originated by user input or environmental events. A load manager 26C facilitates the loading of Blendo files and native node implementations. A media manager 26E provides the ability to load, control and play audio, image and video media assets. A render manager 26G allows the creation and management of objects used to render scenes. A scene manager 26A controls the scene graph. A surface manager 26F allows the creation and management of surfaces onto which scene elements and other assets may be composited. A thread manager 26B gives authors the ability to spawn and control threads and to communicate between them.

Fig. 1B illustrates in a flow diagram, a conceptual description of the flow of content through a Blendo engine. In block 50, a presentation begins with a source which includes a file or stream 34 (Fig. 1A) of content being brought into parser 14 (Fig. 1A). The source could be in a native VRML-like textual format, a native binary format, an XML based format, or the like. Regardless of the format of the source, in block 55, the source is converted into raw scene graph 16 (Fig. 1A). The raw scene graph 16 can represent the nodes, fields and other objects in the content, as well as field initialization values. It also can contain a description of object prototypes, external prototype references in the stream 34, and route statements.

The top level of raw scene graph 16 include nodes, top level fields and functions, prototypes and routes contained in the file. Blendo allows fields and functions at the top level in addition to traditional elements. These are used to provide an interface to an external environment, such as an HTML page. They also provide the object interface when a stream 34 is used as the contents of an external prototype.

Each raw node includes a list of the fields initialized within its context. Each raw field entry includes the name, type (if given) and data value(s) for that field. Each data value includes a number, a string, a raw node, and/or a raw field that can represent an explicitly typed field value.

In block 60, the prototypes are extracted from the top level of raw scene graph 16 (Fig. 1A) and used to populate the database of object prototypes accessible by this scene.

The raw scene graph 16 is then sent through a build traversal. During this traversal, each object is built (block 65), using the database of object prototypes.

In block 70, the routes in stream 34 are established. Subsequently, in block 75, each field in the scene is initialized. This is done by sending initial events to non-default fields of objects. Since the scene graph structure is achieved through the use of node fields, block 75 also constructs the scene hierarchy as well. Events are fired using in-order traversal. The first node encountered enumerates fields in the node. If a field is a node, that node is traversed first.

As a result the nodes in that particular branch of the tree are initialized. Then, an event is sent to that node field with the initial value for the node field.

After a given node has had its fields initialized, the author is allowed to add initialization logic (block 80) to prototyped objects to ensure that the node is fully initialized at call time. The blocks described above produce a root scene. In block 85 the scene is delivered to the scene manager 26A (Fig. 1A) created for the scene. In block 90, the scene manager 26A is used to render and perform behavioral processing either implicitly or under author control.

A scene rendered by the scene manager 26A can be constructed using objects from the Blendo object hierarchy. Appendix B shows the object hierarchy and provides a detailed description of the objects in Blendo. Objects may derive some of their functionality from their parent objects, and subsequently extend or modify their functionality. At the base of the hierarchy is the Object. The two main classes of objects derived from the Object are a Node and a Field. Nodes contain, among other things, a render method, which gets called as part of the render traversal. The data properties of nodes are called fields. Among the Blendo object hierarchy is a class of objects called Timing Objects, which are described in detail below. The following code portions are for exemplary purposes. It should be noted that the line

numbers in each code portion merely represent the line numbers for that particular code portion and do not represent the line numbers in the original source code.

TIMING OBJECTS

Timing objects include a TimeBase node. This is included as a field of a timed node and supplies a common set of timing semantics to the media. Through node instancing the TimeBase node can be used for a number of related media nodes, ensuring temporal synchronization. Blendo also provides a set of nodes including Score which is utilized for sequencing media events. The Score is a timed node and derives its timing from a TimeBase. The Score includes a list of Cue nodes, which emit events at the time specified. Various timing objects, including Score, are described below.

TimedNode

The following code portion illustrates the TimeNode node. A description of the functions in the node follows thereafter.

```
1)  TimedNode : ChildNode {
2)    field TimeBaseNode timeBase NULL
3)    function Time getDuration()
4)    function void updateStartTime(Time now, Time mediaTime, Float rate)
5)    function void updateStopTime(Time now, Time mediaTime, Float rate)
6)    function void updateMediaTime(Time now, Time mediaTime, Float rate)
}
```

This object is the parent of all nodes controlled by a TimeBaseNode.

In line 2 of the code portion, the TimeBase field contains the controlling TimeBaseNode, which makes the appropriate function calls listed below when the time base starts, stops or advances.

In line 3, the getDuration function returns the duration of the TimedNode. If unavailable, a value of -1 is returned. This function is typically overridden by derived objects.

Line 4 lists the updateStartTime function. When called, this function starts advancing its events or controlled media, with a starting offset specified by the mediaTime argument. The updateStartTime function is typically overridden by derived objects.

Line 5 lists the updateStopTime function, which when called, stops advancing its events or controlled media. This function is typically overridden by derived objects.

In line 6, the updateMediaTime function is called whenever mediaTime is updated by the TimeBaseNode. The updateMediaTime function is used by derived objects to exert further control over their media or send additional events.

IntervalSensor

The following code portion illustrates the IntervalSensor node. A description of the fields in the node follows thereafter.

```
1)  IntervalSensor : TimedNode {
2)     field Time  cycleInterval 1
3)     field Float fraction      0
4)     field Float time          0
}
```

The IntervalSensor node generates events as time passes.

IntervalSensor node can be used for many purposes including but not limited to:

- driving continuous simulations and animations;
- controlling periodic activities (*e.g.*, one per minute);
- initiating single occurrence events such as an alarm clock.

The IntervalSensor node sends initial fraction and time events when its updateStartTime() function is called. This node also sends a fraction and time event every time updateMediaTime() is called. Finally, final fraction and time events are sent when the updateStopTime() function is called.

In line 2 of the code portion, the cycleInterval field is set by the author to determine the length of time, measured in seconds, it takes for fraction to go from 0 to 1. This value is returned when the getDuration() function is called.

Line 3 lists the fraction field, which generates events whenever the TimeBaseNode is running using equation (1) below:

fraction = max(min(mediaTime / cycleInterval, 1), 0) Eqn. (1)

Line 4 lists the time field, which generates events whenever the TimeBaseNode is running. The value of the time field is the current wall clock time.

Score

The following code portion illustrates the Score node. A description of the field in the node follows thereafter.

```
1) Score : TimedNode {
2)   field MF CueNode cue [ ]
}
```

This object calls each entry in the cue field for every updateStartTime(), updateMediaTime() and updateStopTime() call received. Calls to each cue entry returns the currently accumulated relative time. This value is passed to subsequent cue entries to allow relative offsets between cue entries to be computed.

In line 2 of the code portion, the cuefield holds the list of CueNode entries to be called with the passage of mediaTime.

TimeBaseNode

The following code portion illustrates the TimeBaseNode node. A description of the fields and functions in the node follows.

```
1)  TimeBaseNode : Node {
2)    field Time mediaTime 0
3)    function void    evaluate(Time time)
4)    function void    addClient(TimedNode node)
5)    function void    removeClient(TimedNode node)
```

```
6)      function Int32    getNumClients()
7)      function TimedNode  getClient(Int32 index)
}
```

This object is the parent of all nodes generating mediaTime.

Line 2 of the code portion lists the mediaTime field, which generates an event whenever mediaTime advances. MediaTime field is typically controlled by derived objects.

Line 3 lists the evaluate function, which is called by the scene manager when time advances if this TimeBaseNode has registered interest in receiving time events.

Line 4 lists addClient function, which is called by each TimedNode when this TimeBaseNode is set in their timeBase field. When mediaTime starts, advances or stops, each client in the list is called. If the passed node is already a client, this function performs no operations.

Line 5 lists the removeClient function, which is called by each TimedNode when this TimeBaseNode is no longer set in their timeBase field. If the passed node is not in the client list, this function performs no operations.

Line 6 lists the getNumClients function, which returns the number of clients currently in the client list.

Line 7 lists the getClient function, which returns the client at the passed index. If the index is out of range, a NULL value is returned.

TimeBase

The following code portion illustrates the TimeBase node. A description of the fields in the node follows thereafter.

```
1)   TimeBase : TimeBaseNode {
2)     field Bool  loop          false
3)     field Time  startTime     0
4)     field Time  playTime      0
5)     field Time  stopTime      0
6)     field Time  mediaStartTime 0
7)     field Time  mediaStopTime  0
```

```
 8)    field Float rate      1
 9)    field Time duration   0
10)    field Bool enabled    true
11)    field Bool isActive   false
}
```

This object controls the advancement of mediaTime. TimeBase can start, stop and resume this value, as well as make mediaTime loop continuously. Time Base allows mediaTime to be played over a subset of its range.

In line 2 of the code portion, the loop field controls whether or not mediaTime repeats its advancement when mediaTime reaches the end of its travel.

In line 3, the startTime field controls when mediaTime starts advancing. When startTime, which is in units of wall clock time, is reached the TimeBase begins running. This is true as long as stopTime is less than startTime. When this occurs mediaTime is set to the value of mediaStartTime if rate is greater than or equal to 0. If mediaStartTime is out of range (see mediaStartTime for a description of its valid range), mediaTime is set to 0. If the rate is less than 0, mediaTime is set to mediaStopTime. If mediaStopTime is out of range, mediaTime is set to duration. The TimeBase continues to run until stopTime is reached or mediaStopTime is reached (mediaStartTime if rate is less than 0). If a startTime event is received while the TimeBase is running, it is ignored.

In line 4 and 5, the playTime field behaves identically to startTime except that mediaTime is not reset upon activation. The playTime field allows mediaTime to continue advancing after the TimeBase is stopped with stopTime. If both playTime and startTime have the same value, startTime takes precedence. If a playTime event is received while the TimeBase is running, the event is ignored. The stopTime field controls when the TimeBase stops.

In line 6, the mediaStartTime field sets the start of the subrange of the media duration over which mediaTime shall run. The range of mediaStartTime is from zero to the end of the duration (0..duration). If the value of mediaStartTime field is out of range, 0 is used in its place.

In line 7, the mediaStopTime field sets the end of the subrange of the media duration over which mediaTime shall run. The range of mediaStopTime is from zero to the end of the duration (0..duration]. If the value of mediaStopTime is out of range, duration is used in its place.

In line 8, the rate field allows mediaTime to run at a rate other than one second per second of wall clock time. The rate provided in the rate field is used as an instantaneous rate. When evaluate is called, the elapsed time since the last call is multiplied by rate and the result is added to the current mediaTime.

In line 9, the duration field generates an event when the duration of all clients of this TimeBase have determined their duration. The value of the duration field is the same as the client with the longest duration.

In line 10, the enabled field enables the TimeBase. When enabled goes false, isActive goes false if it was true and mediaTime stops advancing. While false, startTime and playTime are ignored. When enabled field goes true, startTime and playTime are evaluated to determine if the TimeBase should begin running. If so, the behavior as described in startTime or playTime is performed.

Line 11 lists the isActive field, which generates a true event when the TimeBase becomes active and a false event when the timeBase becomes inactive.

CueNode

The following code snippet illustrates the CueNode node. A description of the fields in the node follows thereafter.

```
1)  CueNode : Node {
2)     field Float offset   -1
3)     field Float delay    0
4)     field Bool  enabled  true
5)     field Int32 direction 0
6)     function void updateStartTime(Time now, Time mediaTime, Float rate)
7)     function void updateStopTime(Time now, Time mediaTime, Float rate)
8)     function Time evaluate(Time accumulated, Time now, Time mediaTime, Float rate)
9)  function Time getAccumulatedTime(Time accumulated)
```

10) function void fire(Time now, Time mediaTime)
}

This object is the parent for all objects in the Score's cue list.

In line 2 of the code portion, the offset field establishes a 0 relative offset from the beginning of the sequence. For instance, a value of 5 will fire the CueNode when the incoming mediaTime reaches a value of 5.

In line 3, the delay field establishes a relative delay before the CueNode fires. If offset is a value other than -1 (the default), this delay is measured from offset. Otherwise the delay is measured from the end of the previous CueNode or from 0 if this is the first CueNode. For instance, if offset has a value of 5 and delay has a value of 2, this node will fire when mediaTime reaches 7. If offset has a value of -1 and delay has a value of 2, this node will fire 2 seconds after the previous CueNode ends.

In line 4, if the enabled field is false, the CueNode is disabled. The CueNode behaves as though offset and delay were their default values and it does not fire events. If it is true, the CueNode behaves normally.

In line 5, the direction field controls how this node fires relative to the direction of travel of mediaTime. If this field is 0, this node fires when this node's offset and/or delay are reached, whether mediaTime is increasing (rate greater than zero) or decreasing (rate less than zero). If direction field is less than zero, this node fires only if its offset and/or delay are reached when mediaTime is decreasing. If direction field is greater than zero, this node fires only if this node's offset and/or delay are reached when mediaTime is increasing.

Line 6 lists the updateStartTime function, which is called when the parent Score receives an updateStartTime() function call. Each CueNode is called in sequence.

Line 7 lists the updateStopTime function, which is called when the parent Score receives an updateStopTime() function call. Each CueNode is called in sequence.

Line 8 lists the evaluate function, which is called when the parent Score receives an updateMediaTime() function call. Each CueNode is called in sequence and must return its

accumulated time. For instance, if offset is 5 and delay is 2, the CueNode would return a value of 7. If offset is -1 and delay is 2, the CueNode would return a value of the incoming accumulated time plus 2. This is the default behavior. Some CueNodes (such as IntervalCue) have a well defined duration as well as a firing time.

In line 9, the getAccumulatedTime function returns the accumulated time using the same calculation as in the evaluate() function.

Line 10 lists the fire function, which is called from the default evaluate() function when the CueNode reaches its firing time. The fire function is intended to be overridden by the specific derived objects to perform the appropriate action.

MediaCue

The following code portion illustrates the MediaCue node. A description of the fields in the node follows thereafter.

```
1)  MediaCue : CueNode TimeBaseNode {
2)     field Time  mediaStartTime 0
3)     field Time  mediaStopTime  0
4)     field Time  duration       0
5)     field Bool  isActive       false
}
```

This object controls the advancement of mediaTime when this CueNode is active. MediaCue allows mediaTime to be played over a subset of its range. MediaCue is active from the time determined by the offset and/or delay field for a length of time determined by mediaStopTime minus mediaStartTime. The value MediaCue returns from getAccumulatedTime() is the value computed by adding the default function to the mediaStopTime and subtracting the mediaStartTime. This node generates mediaTime while active, which is computed by subtracting the firing time plus mediaStartTime from the incoming mediaTime. MediaCue therefore advances mediaTime at the same rate as the incoming mediaTime.

In line 2 of the code portion, the mediaStartTime field sets the start of the subrange of the media duration over which mediaTime runs. The range of mediaStartTime is from zero to

the end of the duration (0..duration). If the value of mediaStartTime field is out of range, 0 is utilized in its place.

In line 3, the mediaStopTime field sets the end of the subrange of the media duration over which mediaTime runs. The range of mediaStopTime is from zero to the end of the duration (0..duration). If the value of mediaStopTime field is out of range, duration is utilized in its place.

In line 4, the duration field generates an event when the duration of all clients of this TimeBaseNode have determined their duration. The value of duration field is the same as the client with the longest duration.

Line 5 lists the isActive field, which generates a true event when this node becomes active and a false event when this node becomes inactive.

IntervalCue

The following code portion illustrates the IntervalCue node. A description of the fields in the node follows thereafter.

```
1)  IntervalCue : CueNode {
2)     field Float period   1
3)     field Bool  rampUp   true
4)     field Float fraction 0
5)     field Bool  isActive false
}
```

This object sends fraction events from 0 to 1 (or 1 to 0 if rampUp is false) as time advances.

Line 2 of the code snippet lists the period field, which determines the time, in seconds, over which the fraction ramp advances.

In line 3, if the rampUp field is true (the default) the fraction goes up from 0 to 1 over the duration of the IntervalCue. If false, the fraction goes down from 1 to 0. If mediaTime is running backwards (when the rate is less than zero), the fraction goes down from 1 to 0 when rampUp field is true, and the fraction goes up from 0 to 1 when the rampUp field is false.

In line 4, the fraction field sends an event with each call to evaluate() while this node is active. If mediaTime is moving forward, fraction starts to output when this node fires and stops when this nodes reaches its firing time plus period. The value of fraction is described as:

fraction = (mediaTime - firing time) * period Eqn. (2)

Line 5 lists the isActive field, which sends a true event when the node becomes active and false when the node becomes inactive. If mediaTime is moving forward, the node becomes active when mediaTime becomes greater than or equal to firing time. This node becomes inactive when mediaTime becomes greater than or equal to firing time plus period. If mediaTime is moving backward, the node becomes active when mediaTime becomes less than or equal to firing time plus period and inactive when mediaTime becomes less than or equal to firing time. The firing of these events is affected by the direction field.

FieldCue

The following code portion illustrates the FieldCue node. A description of the fields in the node follows thereafter.

```
1)  FieldCue : CueNode {
2)     field Field cueValue NULL
3)     field Field cueOut   NULL
}
```

This object sends cueValue as an event to cueOut when FieldCue fires. FieldCue allows any field type to be set and emitted. The cueOut value can be routed to a field of any type. Undefined results can occur if the current type of cueValue is not compatible with the type of the destination field.

In line 2 of the code portion, the cueValue field is the authored value that will be emitted when this node fires.

Line 3 lists the cueOut field, which sends an event with the value of cueValue when this node fires.

TimeCue

The following code portion illustrates the TimeCue node. A description of the field in the node follows thereafter.

```
1)  TimeCue : CueNode {
2)    field Time cueTime  0
}
```

This object sends the current wall clock time as an event to cueTime when TimeCue fires.

Line 2 of the code portion lists the cueTime field, which sends an event with the current wall clock time when this node fires.

The scoring construct within the context of real-time scene composition enables the author to declaratively describe temporal control over a wide range of presentation and playback techniques, including: image flipbooks and image composite animations (e.g., animated GIF); video and audio clips and streams; geometric animation clips and streams, such as joint transformations, geometry morphs, and texture coordinates; animation of rendering parameters, such as lighting, fog, and transparency; modulation of parameters for behaviors, simulations, or generative systems; and dynamic control of asset loading, event routing, and logic functions. For instance, the following example emits a string to pre-load an image asset, then performs an animation using that image, then runs a movie. The string in the following example can also be run in reverse (i.e., first the movie plays backwards then the animation plays backward and then the image disappears).

```
1)  Score {
2)    timeBase DEF TB TimeBase { }
3)    cue [
4)      Fieldcue {
5)        cueValue String " "
6)        cueOut TO ISURF.URL
7)        direction  -1
8)      }
9)      FieldCue {
10)       cueValue String "image1.png"
11)       cueOut TO ISURF.url
12)       direction  -10
```

```
13)          }
14)          IntervalCue {
15)            delay   0.5
16)            period  2.5    # 2.5 second animation
17)            fraction TO PI.fraction
18)          }
19)          DEF MC MediaCue {
20)            offset 2
21)          }
22)          Fieldcue {
23)            cueValue String " "
24)            cueOut TO ISURF.URL
25)            direction  1
26)            delay  -0.5
27)          }
28)          Fieldcue {
29)            cueValue String "image1.png"
30)            cueOut TO ISURF.URL
31)            direction  -1
32)            delay  -0.5
33)          }
34)        ]
35)      }

36)      # Slide out image
37)      DEF T Transform {
38)        children Shape {
39)          appearance Appearance {
40)            texture Texture {
41)              surface DEF ISURF ImageSurface { }
42)            }
43)          }
44)          geometry IndexedFaceSet { ... }
45)        }
46)      }
47)      DEF PI PositionInterpolator {
48)        key ...
49)        keyValue ...
50)        value TO T.translation
51)      }

52)      # Movie
53)      Shape {
54)        appearance Appearance {
55)          texture Texture {
56)            surface MovieSurface {
57)              url "myMovie.mpg"
```

```
58)                 timeBase USE MC
59)               }
60)             }
61)           }
62)           geometry IndexedFaceSet { ... }
63)       }
```

All Cue nodes in a Score fire relative to the media time of the TimeBase, providing a common reference and thereby resulting in an accurate relationship between timing of various media assets. In the code snippet above, the FieldCue (line 9) fires as soon as the TimeBase starts because this FieldCue has default offset and delay fields thereby making the image appear. Lines 35-45 of the code portion loads the image (200, Fig. 2A) on a surface. The IntervalCue (line 13) then starts 0.5 seconds later and runs for the next 2.5 seconds, increasing its fraction output from 0 to 1. The firing of the IntervalCue starts the animation (202, Fig. 2A) of the image. Lines 46-50 control the animation. The MediaCue (line 18) starts 2 seconds after the TimeBase starts, or when the IntervalCue is 1.5 seconds into its animation thereby starting the movie. Lines 51–62 loads the first frame (204, Fig. 2A) of the movie on the surface. When this string is played backwards, first the movie plays in reverse. Then 0.5 seconds later the image appears, and 0.5 seconds after the image appears the animation starts. Animation is played in reverse for 2.5 seconds, when it stops and 0.5 seconds after that the image disappears. This example shows the ability of the Cues to be offset from each other or from the TimeBase and shows that a subsequent Cue can start before the last one has finished.

The MediaCue gives a synchronization tool to the author. A MediaCue is a form of Cue, which behaves exactly like a TimeBase. In fact, a MediaCue can be used where a TimeBase can, as shown in the above example. But since a MediaCue is embedded in a timed sequence of events, an implementation has enough information to request pre-loading on an asset. Fig. 2B illustrates synchronization of the media sequence of Fig. 2A requiring pre-loading. For instance, in the above example, if the implementation knows that a movie takes 0.5 seconds to pre load and play instantly, after waiting (block 210) 1.5 seconds after the start of the TimeBase, in block 215, a "get ready" signal is sent to the MovieSurface. Upon receipt of get ready signal, in block 220 the movie is pre-loaded. This would give it the required 0.5

seconds to pre-load. In block 225 a request to start is received, and upon receipt of the request to start, block 230 starts the movie instantly.

The combination of the TimeBase and media sequencing capabilities allowed in Blendo makes it possible to create presentations with complex timing. Fig. 3 shows time relationships of various components of a Blendo presentation. A viewer, upon selecting news presentation (360), sees a screen wherein he can select a story (362). Upon the user selecting story S3 from a choice of five stories S1, S2, S3, S4 and S5, a welcome screen with an announcer is displayed (364). On the welcome screen the viewer can choose to switch to another story (374) thereby discontinuing story S3. After the welcome statement, the screen transitions to the site of the story (366) and the selected story is played (368). At this point, the viewer can go to the next story, the previous story, rewind the present story or select to play an extended version of story (370) S3 or jump to (372), for example, another story S5. After the selected story is played the user can make the next selection.

It is to be understood that the present invention is independent of Blendo, and it can be part of an embodiment separate from Blendo. It is also to be understood that the present invention is equally applicable to 2D scene rendering and 3D scene rendering.

While particular embodiments of the present invention have been described it will be apparent to those skilled in the art that changes and modifications may be made without departing from this invention in its broader aspect and, therefore, the appended claims are to encompass within their scope all such changes and modifications as fall within the true sprit and scope of this invention.